PCS 3115

Sistemas Digitais I

Circuitos Combinatórios

Blocos Básicos:

(De)Codificadores

Prof. Dr. Marcos A. Simplicio Jr.

versão: 3.0 (Jan/2016)

Adaptado por Glauber (2018)

Blocos básicos

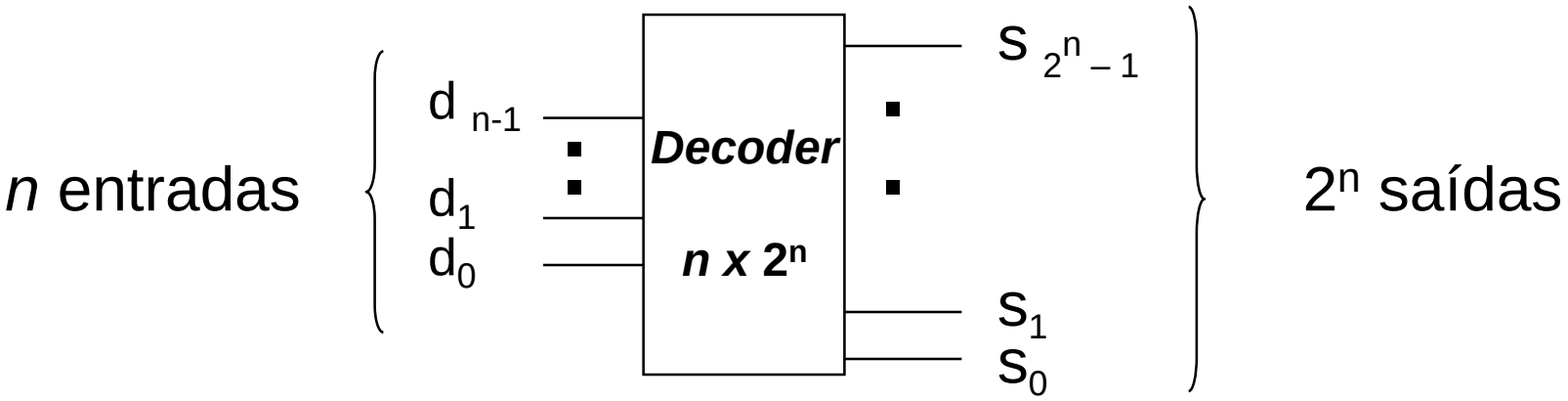
- Codificadores e Decodificadores (HOJE)
- (De) Multiplexadores
- Portas tri-state
- Comparadores
- Somadores/Subtratores
- **16 de maio: P2**
- Multiplicadores
- ULA
- Gerador/Detector de Paridade

Contexto

- Circuitos SSI: *small scale integration*
  - Até ~20 transistores
  - Circuitos bastante simples, como portas lógicas NOT e AND, OR, XOR de 3 entradas
- Circuitos MSI: *medium scale integration*
  - Até ~200 transistores
  - São os circuitos que veremos neste módulo: codificadores, multiplexadores, somadores, etc.

Decodificador

- Converte código de entrada em código de saída
  - Bloco combinatório lógico que possui n entradas e (até) 2^n saídas
  - Tipo comum: codificador para código 1-de-m
    - Para cada combinação de valores das n entradas, **apenas uma saída é ativada**

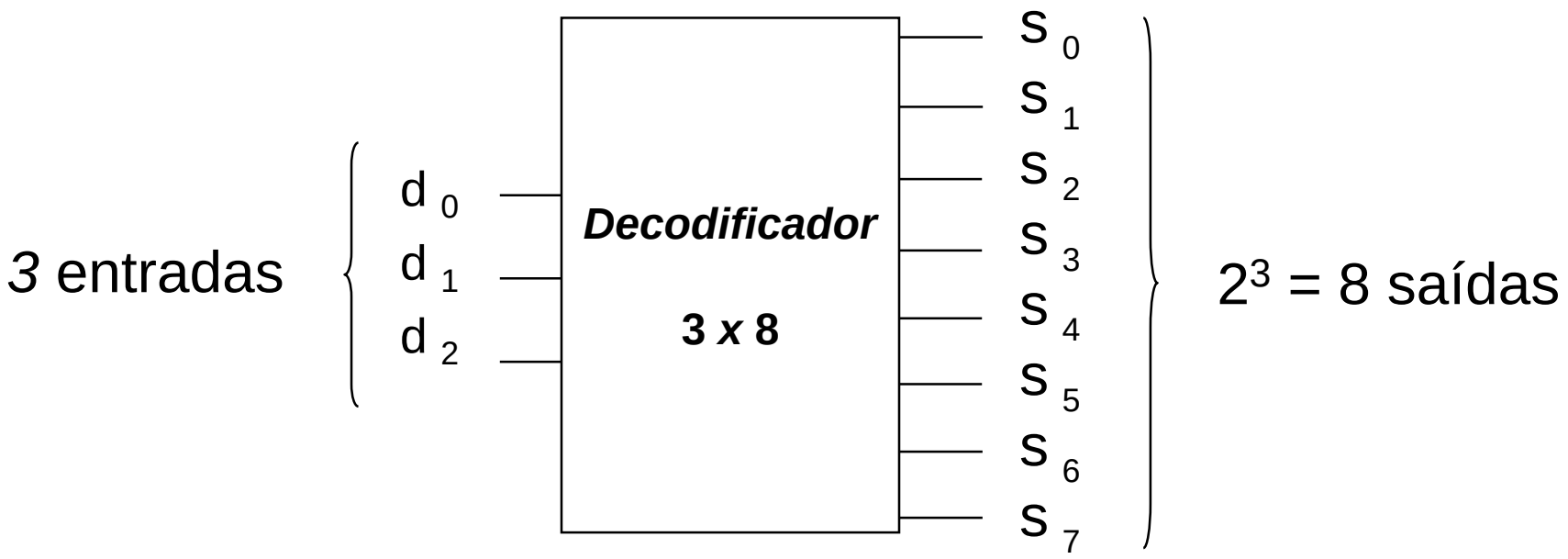


Decodificador 1-de-m

- Funcionamento:
  - Entradas formam uma palavra binária de n bits
  - Palavra de entrada pode assumir os valores de 0 a $2^n - 1$
  - As 2^n saídas S_i são indexadas de 0 a $2^n - 1$
  - Saída ativada S_i é aquela cujo índice i corresponde ao valor da palavra binária de entrada
- Exemplo
  - Entradas = 011 (3_{10}); ativa-se a saída S_3
  - Entradas = 101 (5_{10}); ativa-se a saída S_5

Decodificador de 3 bits

- Símbolo funcional (convenção):
  - Entradas à esquerda
  - Saídas à direita
  - Índice menor indica bit menos significativo na palavra binária



Decodificador de 3 bits

- Tabela verdade

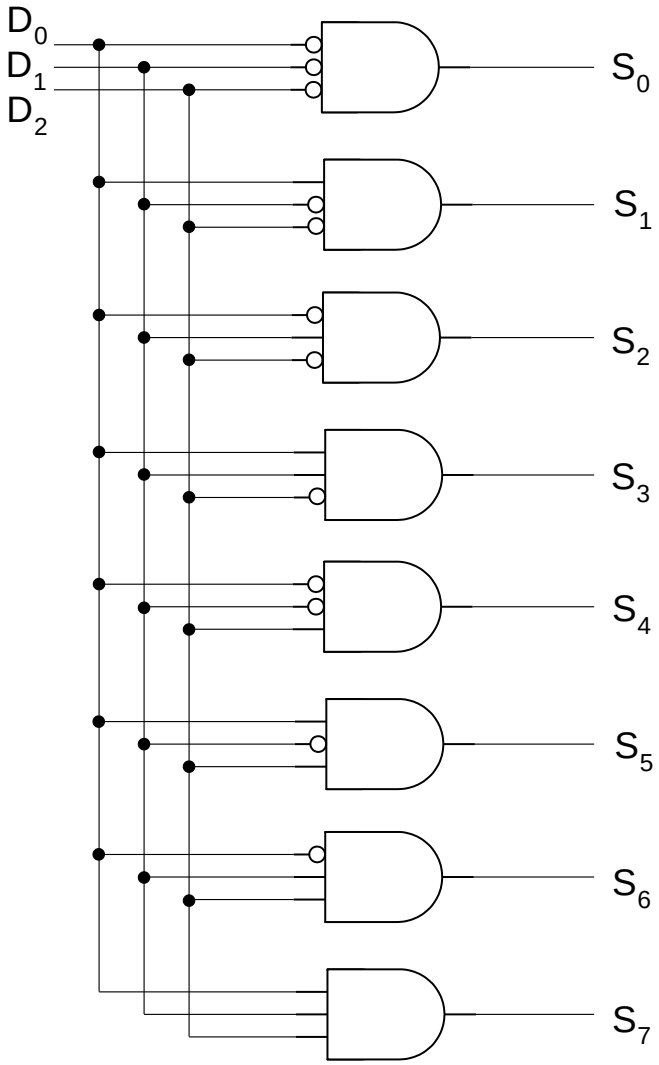
Saída S_2 é ativada...

2 em decimal

ENTRADAS			SAÍDAS							
d_2	d_1	d_0	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0	0	**1**
0	0	1	0	0	0	0	0	0	**1**	0
0	1	0	0	0	0	0	0	**1**	0	0
0	1	1	0	0	0	0	**1**	0	0	0
1	0	0	0	0	0	**1**	0	0	0	0
1	0	1	0	0	**1**	0	0	0	0	0
1	1	0	0	**1**	0	0	0	0	0	0
1	1	1	**1**	0	0	0	0	0	0	0

Decodificador de 3 bits

Circuito interno: mintermos!



$S_0 = D_2' . D_1' . D_0'$

$S_1 = D_2' . D_1' . D_0$

$S_2 = D_2' . D_1 . D_0'$

$S_3 = D_2' . D_1 . D_0$

$S_4 = D_2 . D_1' . D_0'$

$S_5 = D_2 . D_1' . D_0$

$S_6 = D_2 . D_1 . D_0'$

$S_7 = D_2 . D_1 . D_0$

$S_i = m_i$

Decodificador com Enable

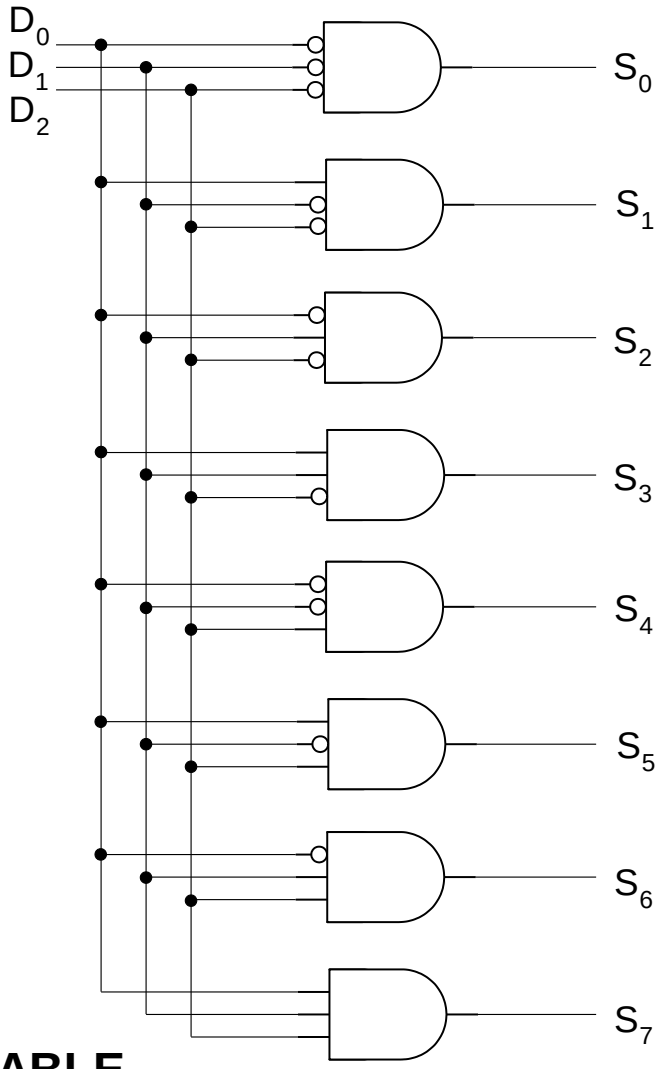
- Entrada adicional: Enable
  - Permite habilitar/desabilitar o bloco todo
- SE Enable ativo
  - Funcionamento normal: apenas uma saída ativa
- SE Enable inativo
  - Nenhuma saída ativa, independente do código nas entradas de endereço
- Atua em todos os mintermos

Decodificador com Enable

Ex: Decodificador 3 por 8, com ***ENABLE*** (EN)

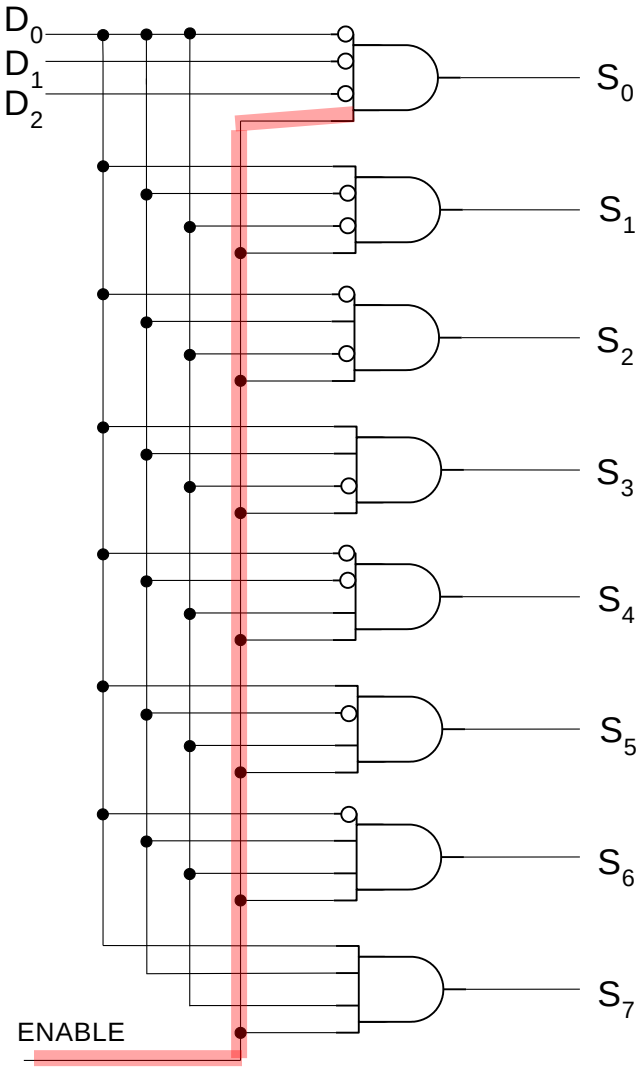
ENTRADAS				SAÍDAS							
EN	d_2	d_1	d_0	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
1	0	0	0	0	0	0	0	0	0	0	**1**
1	0	0	1	0	0	0	0	0	0	**1**	0
1	0	1	0	0	0	0	0	0	**1**	0	0
1	0	1	1	0	0	0	0	**1**	0	0	0
1	1	0	0	0	0	0	**1**	0	0	0	0
1	1	0	1	0	0	**1**	0	0	0	0	0
1	1	1	0	0	**1**	0	0	0	0	0	0
1	1	1	1	**1**	0	0	0	0	0	0	0
0	X	X	X	0	0	0	0	0	0	0	0

Decodificador com Enable



Como adicionar o enable?

Decodificador com Enable

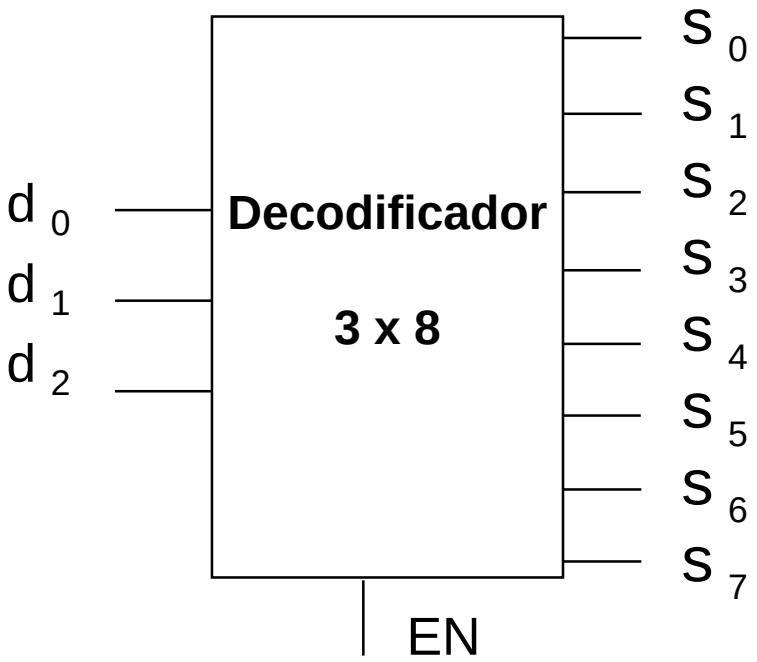


$S_0 = EN \cdot D_2' \cdot D_1' \cdot D_0'$

.

..

$S_7 = EN \cdot D_2 \cdot D_1 \cdot D_0$



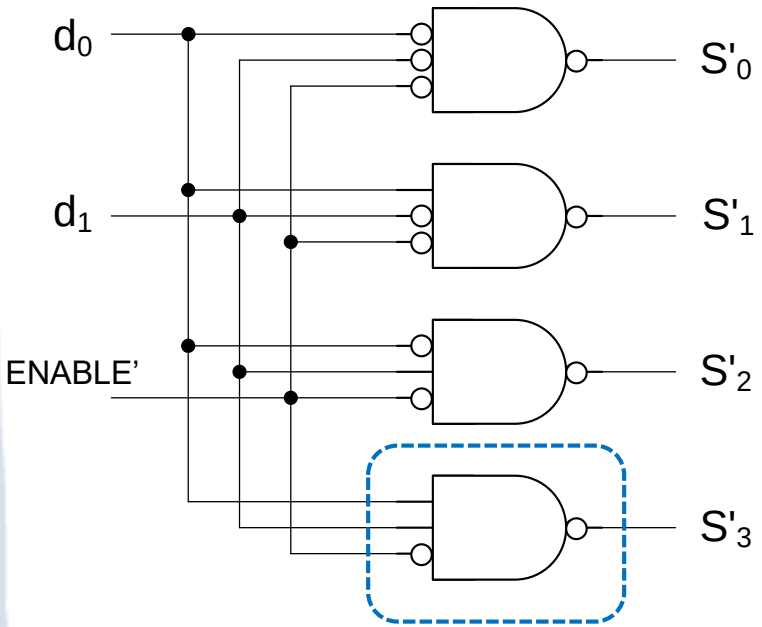
Decodificador: active-low

- Decodificadores podem ter as saídas invertidas (complementadas)
  - Diz-se que as saídas são *active-low* – ativas em ZERO
  - Logo, saídas inativas = 1; a única saída ativa = 0
- Implicações:
  - Na tabela verdade: inversão nas saídas
  - No circuito: uso de **NAND** no lugar de AND
  - Nomenclatura: S_i'
  - Cada saída é um **maxtermo** dos bits de entrada

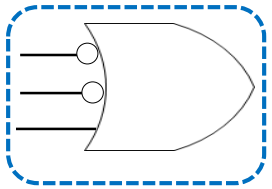
Decodificador: active-low

- Entrada ENABLE também pode ser *active-low*
- SE Enable'=0, circuito habilitado
- SE Enable'=1, saídas todas desabilitadas
- Implicações
  - Na tabela verdade: inversão na entrada
  - No circuito: uso de inversor
  - Nomenclatura: EN'

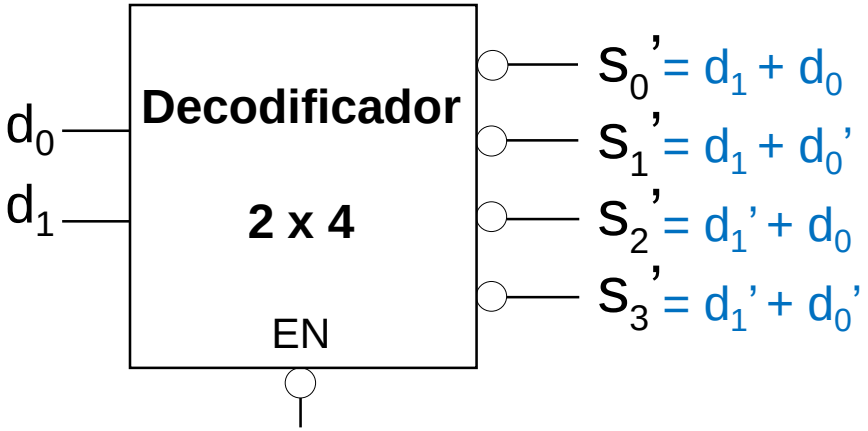
Decodificador: active-low



=



$S_3' = d_1' + d_0'$

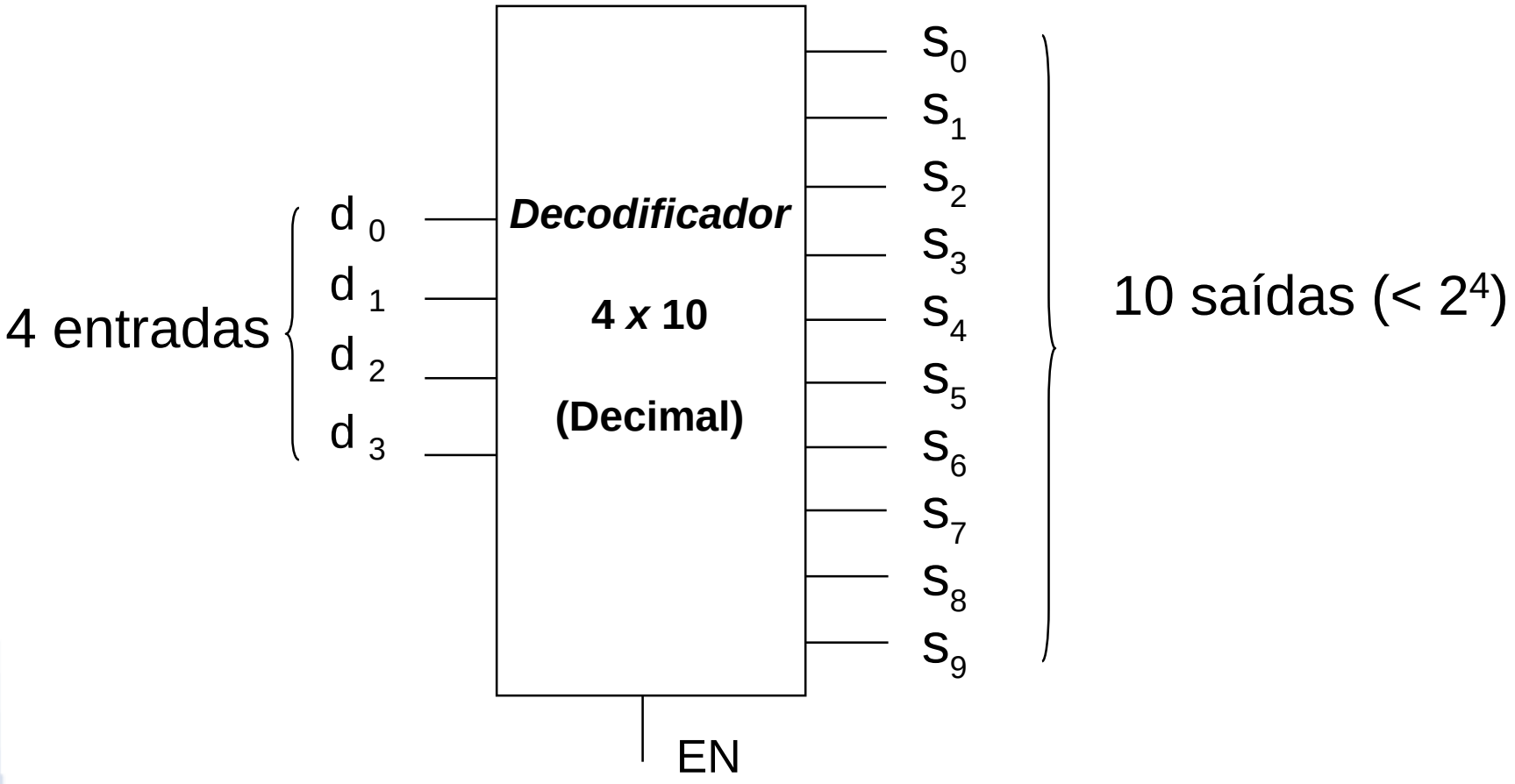


Entradas			Saídas			
EN'	d_1	d_0	S_3'	S_2'	S_1'	S_0'
0	0	0	1	1	1	**0**
0	0	1	1	1	**0**	1
0	1	0	1	**0**	1	1
0	1	1	**0**	1	1	1
1	X	X	1	1	1	1

Decodificadores Decimais

- Decodificador BCD (Binary Coded Decimal) para decimal
  - Entrada: 1 dígito BCD (4 bits)
    - Palavras válidas: 0000 a 1001 (0 a 9)
  - Saída: 10 saídas ($<2^4$)
    - Saída ativa: corresponde à palavra da entrada
    - Saídas são todas inativas se:
      - Enable = Falso OU
      - Palavra de entrada > 1001 (inválida)

Decodificadores Decimais



Decodificadores Decimais

EN	d_3	d_2	d_1	d_0	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
1	0	0	0	1	0	0	0	0	0	0	0	0	**1**	0
1	0	0	1	0	0	0	0	0	0	0	0	**1**	0	0
1	0	0	1	1	0	0	0	0	0	0	**1**	0	0	0
1	0	1	0	0	0	0	0	0	0	**1**	0	0	0	0
1	0	1	0	1	0	0	0	0	**1**	0	0	0	0	0
1	0	1	1	0	0	0	0	**1**	0	0	0	0	0	0
1	0	1	1	1	0	0	**1**	0	0	0	0	0	0	0
1	1	0	0	0	0	**1**	0	0	0	0	0	0	0	0
1	1	0	0	1	**1**	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
	.	.	.	.										
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	X	X	X	X	0	0	0	0	0	0	0	0	0	0

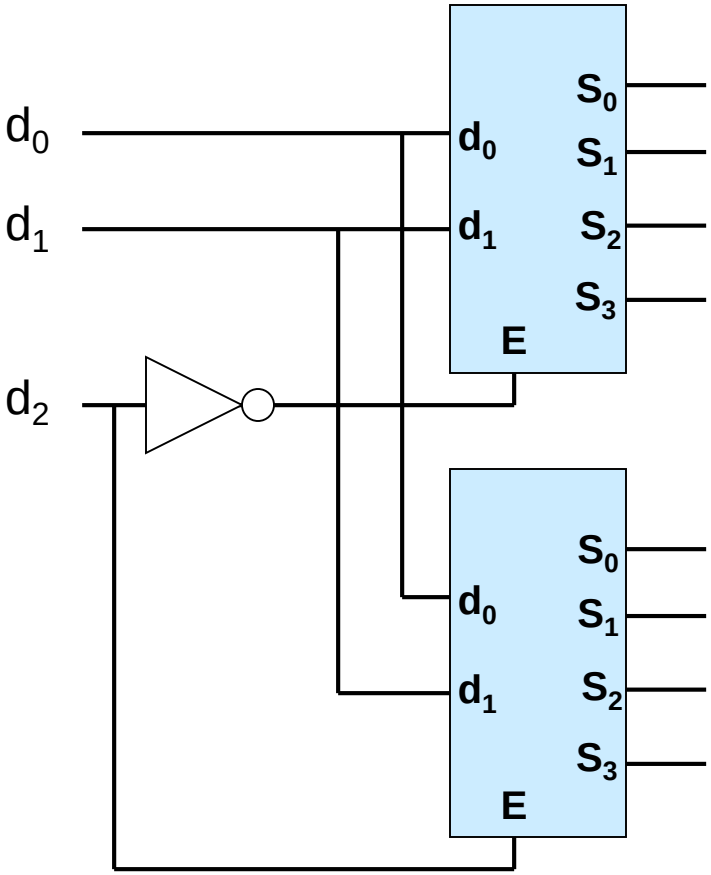
Decodificadores: cascateamento

- Objetivo: expansão de capacidade:
  - Decodificadores 2x4 ➔ 3x8 ➔ 4x16 ➔ 5x32 ➔ n x 2^n
- Estratégia: combinar decodificadores menores, usando entrada de habilitação

- **Exemplo**: montar decodificador 3x8 sem entrada de habilitação a partir de dois decodificadores 2x4 com entrada de habilitação

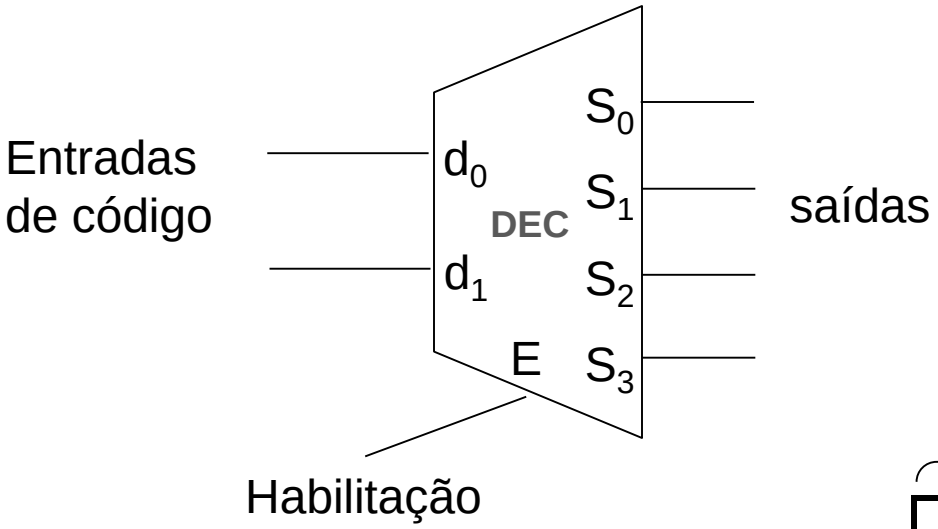
Decodificadores: cascateamento

- Decodificador 3x8, sem entrada de habilitação



Exercício: como adicionar uma entrada de habilitação neste circuito?

Decodificadores: Símbolo alternativo

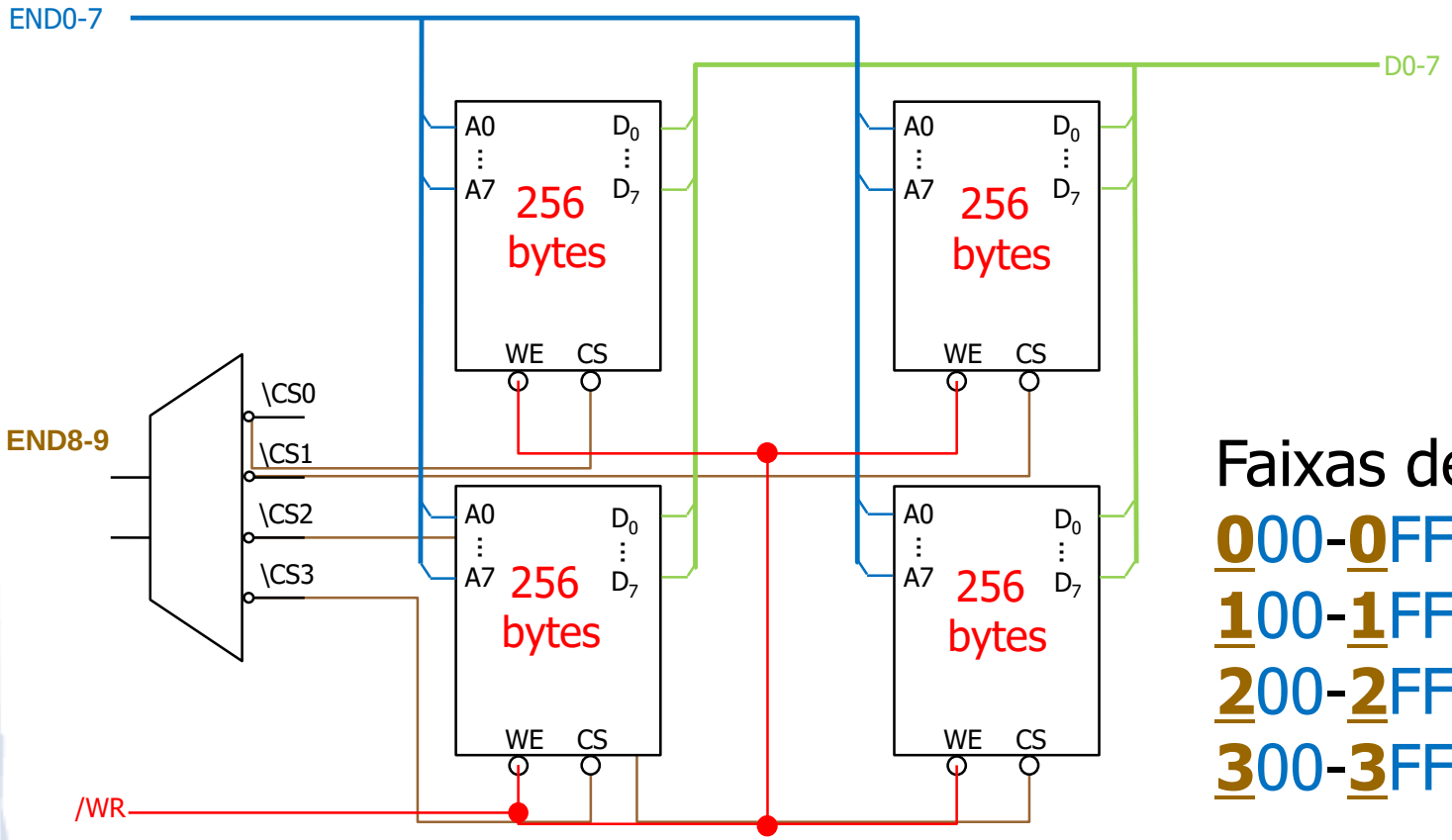


Entradas			Saídas			
E	**d_1**	**d_0**	**S_3**	**S_2**	**S_1**	**S_0**
0	X	X	0	0	0	0
1	0	0	0	0	0	**1**
1	0	1	0	0	**1**	0
1	1	0	0	**1**	0	0
1	1	1	**1**	0	0	0

Tabela verdade

Decodificador: uso (memórias)

Exercício: Dada 4 memórias RAM 256 x 1 byte, obter uma de 1kB.



Faixas de endereço

000-0FF h

100-1FF h

200-2FF h

300-3FF h

Decodificadores: uso (síntese)

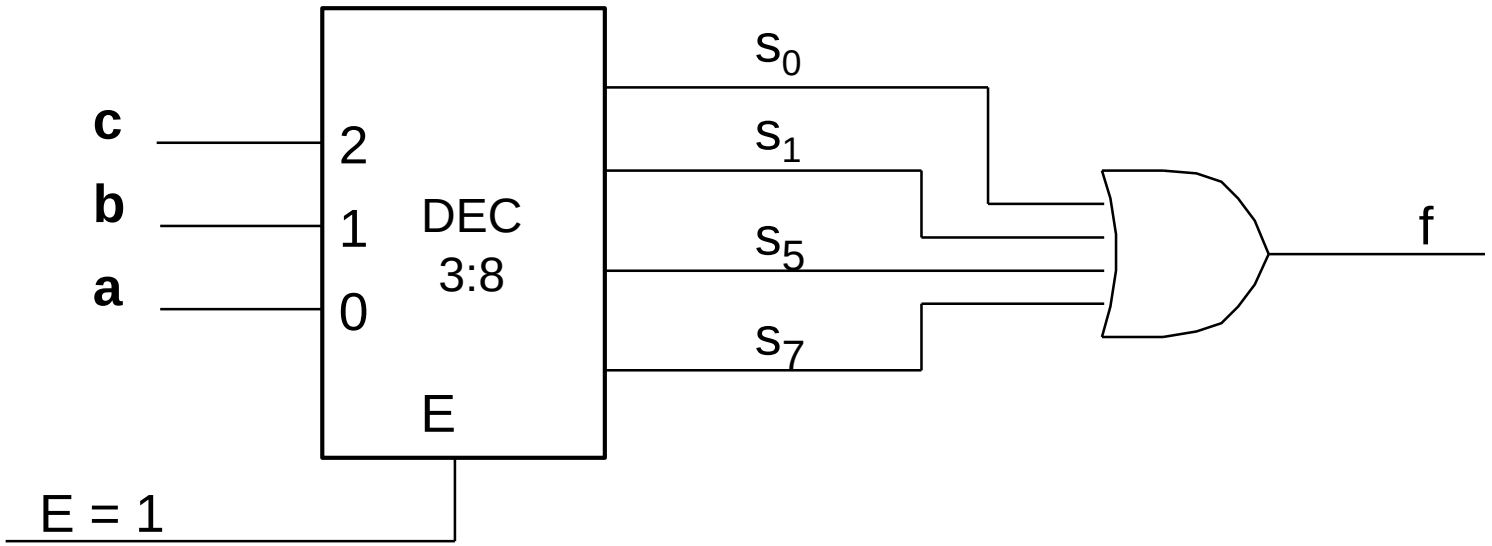
- Síntese de funções de chaveamento
  - Saídas dos decodificadores são os mintermos para as variáveis de entrada
  - 1ª forma canônica: Função = $\Sigma_{mintermos}$

- **Exemplo**:
  - F(d,c,b,a) = Σ(0,1,5,9,12)

Decodificadores: uso (síntese)

- **Exemplo: ativo-alto ➔ soma de mintermos**

  F(c,b,a) = Σ(0,1,5,7)

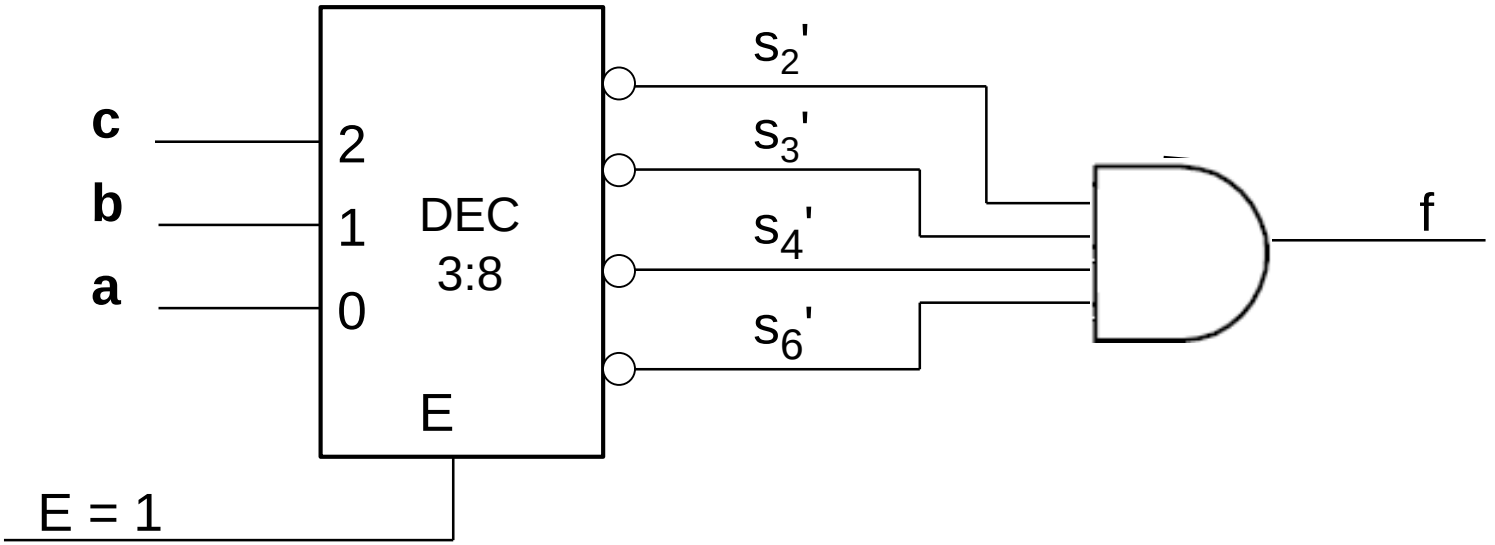


Decodificadores: uso (síntese)

- **Exemplo: ativo-baixo ➔ produto de maxtermos**

$$F(c,b,a) = \Sigma(0,1,5,7)$$

$$= \prod(2,3,4,6)$$



Decodificador 3 x 8: VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity decoder is
    port (d : in STD_LOGIC_VECTOR (2 downto 0);
          s : out STD_LOGIC_VECTOR (7 downto 0));
end decoder;

architecture decoder_arch of decoder is
begin
    with d select
        s <=  "00000001" when "000",
              "00000010" when "001",
              "00000100" when "010",
              "00001000" when "011",
              "00010000" when "100",
              "00100000" when "101",
              "01000000" when "110",
              "10000000" when "111",
              "00000000" when others;
end decoder_arch;
```

Decodificador 3 x 8 com EN: VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity decoder is
    port (d    : in STD_LOGIC_VECTOR (2 downto 0);
          en   : in STD_LOGIC;
          s    : out STD_LOGIC_VECTOR (7 downto 0));
end decoder;

architecture decoder_arch of decoder is
begin
    s <= "00000000" when (en = '0')   else   -- com enable
         "00000001" when (d = "000")  else
         "00000010" when (d = "001")  else
         "00000100" when (d = "010")  else
         "00001000" when (d = "011")  else
         "00010000" when (d = "100")  else
         "00100000" when (d = "101")  else
         "01000000" when (d = "110")  else
         "10000000" when (d = "111")  else
         "00000000";
end decoder_arch;
```

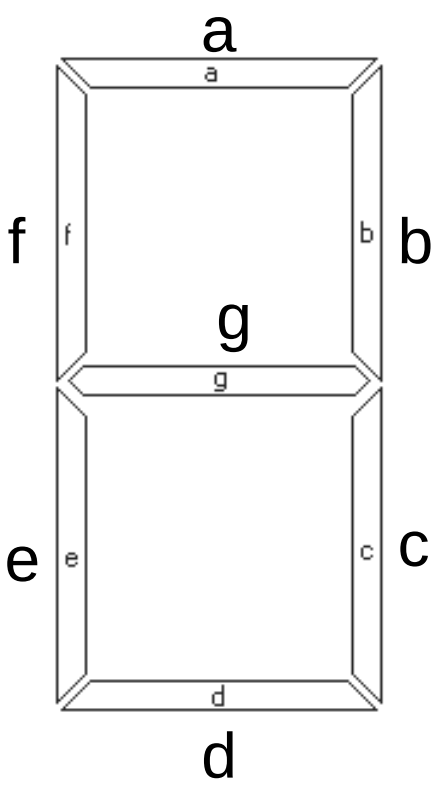
Decodificadores de 7 segmentos

- Decodificadores para acionamento de *displays* de 7 segmentos
  - de BCD para 7 segmentos
  - de binário para 7 segmentos
- Entradas: 4 bits
- Saídas: 7 bits, código 7 segmentos

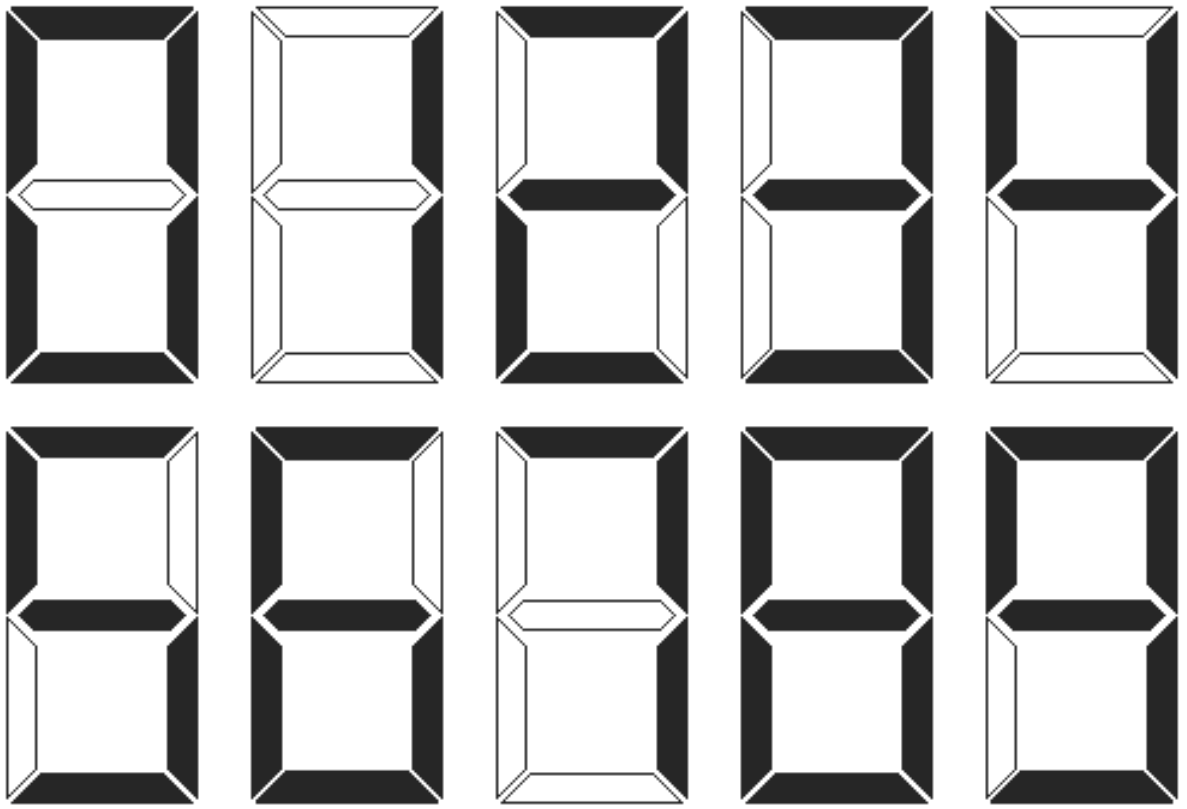
Codificadores

Display de 7 segmentos

Os sete segmentos



O traçado dos algarismos



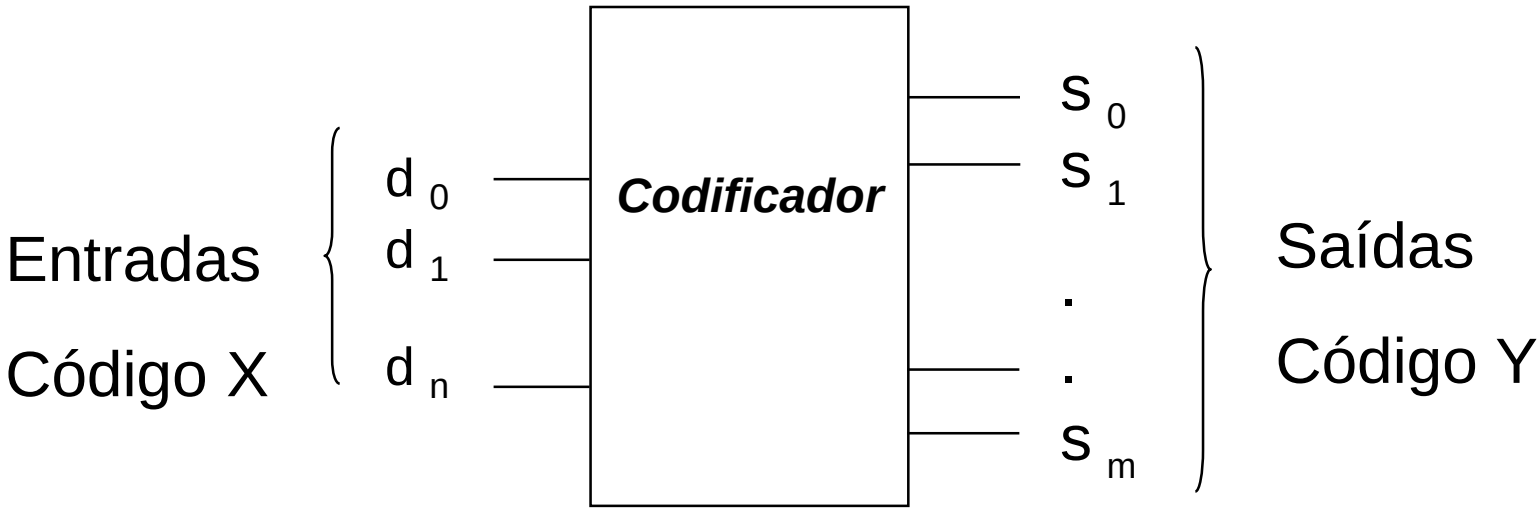
Decodificadores: BCD – 7 segmentos

Tabela verdade

Decimal	EN	d_3	d_2	d_1	d_0	a	b	c	d	e	f	g
0	1	0	0	0	0	1	1	1	1	1	1	
1	1	0	0	0	1		1	1				
2	1	0	0	1	0	1	1		1	1		1
3	1	0	0	1	1	1	1	1	1			1
4	1	0	1	0	0		1	1			1	1
5	1	0	1	0	1	1		1	1		1	1
6	1	0	1	1	0	1		1	1	1	1	1
7	1	0	1	1	1	1	1	1				
8	1	1	0	0	0	1	1	1	1	1	1	1
9	1	1	0	0	1	1	1	1	1		1	1
10	1	1	0	1	0	0	0	0	0	0	0	0
		.	.	.	.							
15	1	1	1	1	1	0	0	0	0	0	0	0
	0	X	X	X	X	0	0	0	0	0	0	0

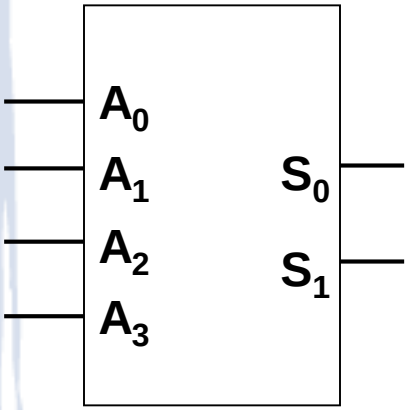
Codificadores

- Codificadores
  - Geram um código específico: transformam um código (maior) em outro (menor)
  - Ex.: código 1 entre $2^n \rightarrow$ código binário

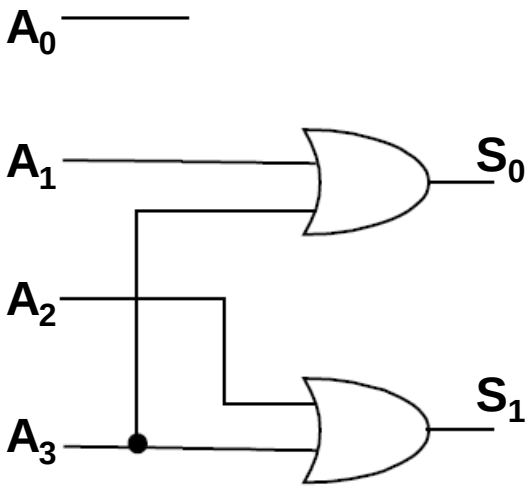


Codificadores: Binário "$2^n : n$"

- Codificador binário 4:2
- Problema em potencial:
  - Mais de uma entrada em 1 pode resultar em erro
  - Ex.: 0110 → 11 (mesmo resultado obtido com 1000)



Entradas				Saídas	
A_3	A_2	A_1	A_0	S_1	S_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	1	1	0	1	1



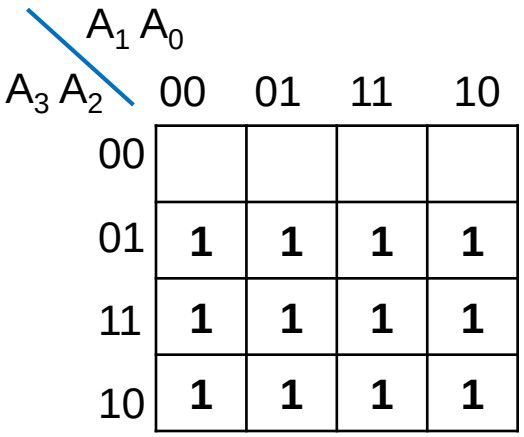
Codificadores: Binário "2^n : n"

- Codificador binário 4:2 **com prioridade**
  - Entradas têm níveis de prioridade: elimina o problema anterior (perceba os "don't care")
  - Resolução: construir o mapa de Karnaugh

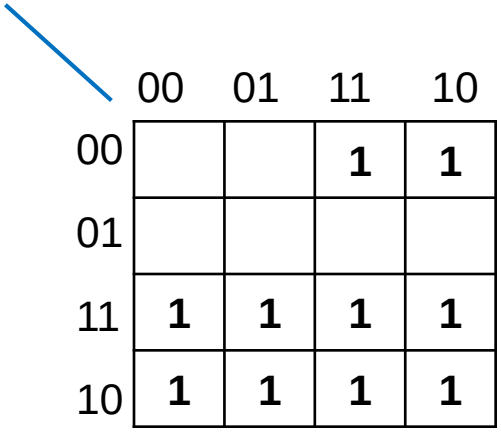
Entradas | Saídas

A_3	A_2	A_1	A_0	S_1	S_0
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1
0	1	1	0	1	0

prioridade



$S_1 = A_3 + A_2$



$S_0 = A_3 + A_1 \bullet A'_2$

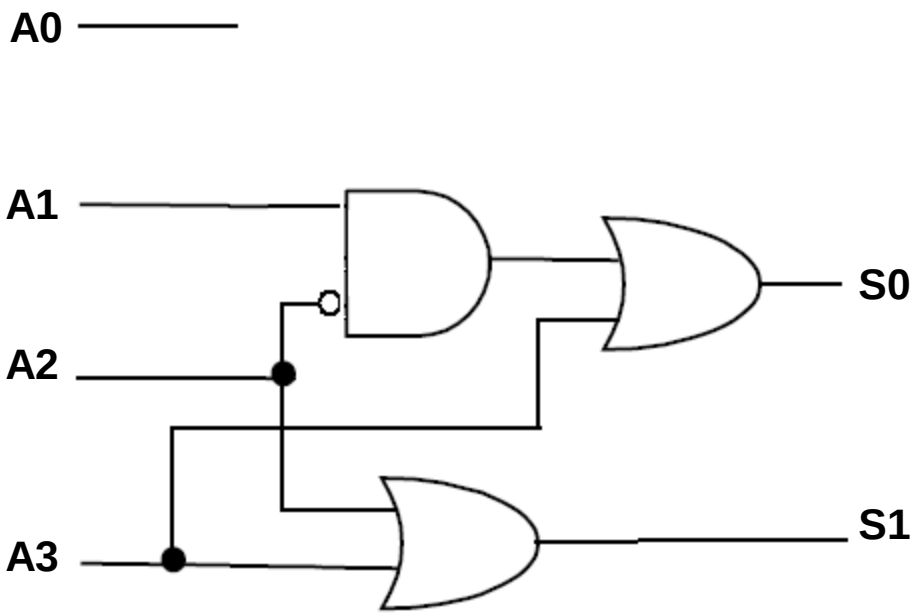
Codificadores: Binário “$2^n : n$”

- Codificador binário 4:2 **com prioridade**
  - Entradas têm níveis de prioridade: elimina o problema anterior (perceba os “don’t care”)
  - Usando o mapa de Karnaugh

Entradas | Saídas

A_3	A_2	A_1	A_0	S_1	S_0
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1
0	1	1	0	1	0

prioridade



Codificadores: uso

- Reduzem o número de fios e aplicações com diversas entradas
  - Ex.: codificador de teclado (em vez de ter um fio por tecla, valor de cada tecla pode ser antes codificado)
- Regular entradas com diferentes prioridades
  - Ex.: controle de interrupções em computadores
- Conversão entre sistemas numéricos
  - Ex.: o codificador BCD (Binary Coded Decimal – Decimal Codificado em Binário) apresenta 10 entradas, resultando em 4 bits de saída em binário: A5 → 0101, A8 → 1000, etc.

Codificadores: VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity priority_encoder is
    port (A    : in STD_LOGIC_VECTOR (7 downto 0);
          en   : in STD_LOGIC;
          S    : out STD_LOGIC_VECTOR (2 downto 0));
end priority_encoder;

architecture arch of priority_encoder is
begin
    S <= "000" when (en   = '0')  else  -- com enable
         "111" when (A(7) = '1')  else  -- maior prioridade
         "110" when (A(6) = '1')  else
         "101" when (A(5) = '1')  else
         "100" when (A(4) = '1')  else
         "011" when (A(3) = '1')  else
         "010" when (A(2) = '1')  else
         "001" when (A(1) = '1')  else
         "000" when (A(0) = '1')  else -- menor prioridade
         "000";
end arch;
```

Lição de Casa

- Leitura sugerida:
  - Capítulo 6 do Livro Texto, ênfase em 6.4 e 6.5.
- Exercícios sugeridos:
  - Capítulo 6 do Livro Texto - (De)Codificadores.